Employee.java

```
/**
 * class: Employee
 * purpose: Model a single employee
 */

public class Employee {

  private int    id;
  private int    age;
  private int    duties;
  private char   gender;
  private String name;

  /**
   * Default constructor
   */

  public Employee () {
    id     = -1;
    age    = -1;
    duties = -1;
    gender = 'm';
    name   = "";
  }

  /**
   * Alternative constructor
   * @param _id - employee id
   * @param _name - employee name
   * @param _gender - employee gender
   * @param _age - employee age
   * @param _duties - employee duties
   */

  public Employee (int _id, String _name, char
_gender, int _age, int _duties) {
    id     = _id;
    name   = _name;
    gender = _gender;
    age    = _age;
    age    = _age;
    duties = _duties;
  }

  /**
   * Convert the employee into an array of
Strings
   * @return String[] - the employee details
serialized
   */

public String[] serialize () {
  return new String[] {
    Integer.toString(id),
    name,
    Character.toString(gender),
    Integer.toString(age),
    Integer.toString(duties)
  };
}

/**
 * Get id
 * @return id
 */

public int getId () {
  return this.id;
}

/**
 * Set id
 * @param id
 */

public void setId (int _id) {
  this.id = _id;
}

/**
 * Get age
 * @return age
 */

public int getAge () {
  return this.age;
}

/**
 * Set age
 * @param age
 */

public void setAge (int _age) {
  this.age = _age;
}

/**
 * Get duties
 * @return duties
 */

public int getDuties () {
  return this.duties;
}

  /**
   * Set duties
   * @param duties
   */

  public void setDuties (int _duties) {
    this.duties = _duties;
  }

  /**
   * Get gender
   * @return gender
   */

  public char getGender () {
    return this.gender;
  }

  /**
   * Set gender
   * @param gender
   */

  public void setGender (char _gender) {
    this.gender = _gender;
  }

  /**
   * Get name
   * @return name
   */

  public String getName () {
    return this.name;
  }

  /**
   * Set name
   * @param name
   */

  public void setName (String _name) {
    this.name = _name;
  }

}
```

Employees.java

```
/**
 * class: Employees
 * purpose: Handle a collection of employees,
including methods to save and load the
collection to a file.
 */

import java.util.ArrayList;
import java.util.Iterator;
import java.io.File;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.PrintWriter;
import java.io.FileWriter;
import java.io.FileNotFoundException;
import java.io.IOException;

public class Employees {

  /**
   * Private variables
   */

  private final String SAVEFILE =
"employees.txt";
  private final String ERROR_OPEN = "File %s
cannot be opened\n";
  private final String ERROR_READ = "File %s
cannot be read\n";

  private ArrayList<Employee> employees;
  private int nextId = 101;
  private final int max = 10;

  /**
   * Public variables
   */

  public int length;


  /**
   * Default constructor
   */

  public Employees () {
    this.length = 0;
    this.employees = new ArrayList<Employee>();
  }
```

```
  /**
   * Add employee to the records
   * @param employee
   * @return id
   */

  public int add (Employee employee) {
    int id = employee.getId();
    if (id == -1) {
      id = this.nextId++;
      employee.setId(id);
    } else {
      this.nextId = id + 1;
    }
    this.employees.add(employee);
    this.length = this.employees.size();
    return id;
  }


  /**
   * Check if an employee with id exists
   * @param id
   * @return boolean
   */

  public boolean has (int id) {
    Employee employee;
    for (int i = 0; i < this.length; i++) {
      employee = this.employees.get(i);
      if (employee.getId() == id) {
        return true;
      }
    }
    return false;
  }


  /**
   * Get an employee by id
   * @param id
   * @return Employee
   */

  public Employee get (int id) {
    Employee employee;
    for (int i = 0; i < this.length; i++) {
      employee = this.employees.get(i);
      if (employee.getId() == id) {
        return employee;
      }
    }
    throw new IndexOutOfBoundsException();
  }
```

```
  /**
   * Get an employee by index
   * @param index
   * @return Employee
   */

  public Employee at (int index) {
    return this.employees.get(index);
  }

  /**
   * Remove an employee from the records
   * @param id
   */

  public void remove (int id) {
    for (int i = this.length - 1; i >= 0; i--)
{
      if (this.at(i).getId() == id) {
        this.employees.remove(i);
        break;
      }
    }
    this.length = this.employees.size();
  }


  /**
   * Retrieve a list of all the Employees in a
certain age group
   * @param min - minimum age (inclusive)
   * @param max - maximum age (inclusive)
   * @return new Employees
   */

  public Employees inAgeGroup (int min, int
max) {
    Employees list = new Employees();
    Employee employee;
    int age;

    for (int i = 0; i < this.length; i++) {
      employee = this.employees.get(i);
      age = employee.getAge();
      if (age >= min && age <= max) {
        list.add(employee);
      }
    }

    return list;
  }
```

```
  /**
   * Get the employee with the most jobs
   */

  public Employee withMostDuties () {
    Employee employee = null;
    int max = 0;
    for (int i = 0; i < this.length; i++) {
      if (this.employees.get(i).getDuties() >
max) {
        employee = this.employees.get(i);
        max = employee.getDuties();
      }
    }
    return employee;
  }


  /**
   * Get a list of the employees with no duties
   */

  public Employees withNoDuties () {
    Employees list = new Employees();
    Employee employee;
    int duties;

    for (int i = 0; i < this.length; i++) {
      employee = this.employees.get(i);
      duties = employee.getDuties();
      if (duties == 0) {
        list.add(employee);
      }
    }

    return list;
  }


  /**
   * Write records to disk
   */

  public void write () {
    FileWriter file = null;
    PrintWriter writer = null;

    try {
      file = new FileWriter(SAVEFILE);
      writer = new PrintWriter(file);
    }
    catch (FileNotFoundException e) {
      this.error(String.format(ERROR_OPEN,
SAVEFILE));
    }
    catch (IOException e) {
      this.error(String.format(ERROR_READ,
SAVEFILE));
    }

    // Loop through the ArrayList of employees
    Employee employee;
    for (int i = 0; i < this.length; i++) {
      // Write each employee to the file
      employee = this.employees.get(i);
      writer.println(this.stringify(employee));
    }

    writer.close();
  }


  /**
   * Read records from disk
   */

  public void read () {
    try {
      FileReader file = new
FileReader(SAVEFILE);
      BufferedReader reader = new
BufferedReader(file);
      String line = reader.readLine();
      while (line != null) {
        this.add(this.parse(line));
        line = reader.readLine();
      }
    }
    catch (FileNotFoundException e) {
      this.error(String.format(ERROR_OPEN,
SAVEFILE));
    }
    catch (IOException e) {
      this.error(String.format(ERROR_READ,
SAVEFILE));
    }
  }

  /**
   * Check if the save file exists
   * @return boolean
   */

  public boolean fileExists () {
    return new File(SAVEFILE).exists();
  }


  /**
   * Check if records are full
   * @return boolean
   */

  public boolean isFull () {
    return this.max <= this.length;
  }


  /**
   * Convert records into an array of arrays of
Strings
   * @return rows
   */

  public String[][] serialize () {
    String[][] rows = new String[this.length]
[5];
    for (int i = 0; i < this.length; i++) {
      rows[i] =
this.employees.get(i).serialize();
    }
    return rows;
  }


  /**
   * Print error and exit
   * @param message - error message
   */

  private void error (String message) {
    System.out.println("Fatal error: " +
message);
    System.exit(0);
  }


  /**
   * Convert employee into string
   * @return string
   */

  private String stringify (Employee employee)
{
    String string = "";
    string += employee.getId()     + ",";
    string += employee.getName()   + ",";
    string += employee.getGender() + ",";
    string += employee.getAge()    + ",";
    string += employee.getDuties();
    return string;
```

```
  }


  /**
   * Convert string into an employee
   * @return Employee
   */

  private Employee parse (String string) {
    String[] parts = string.split(",");
    Employee employee = new Employee();

    if (parts.length < 5) {
      throw new
UnsupportedOperationException();
    }


employee.setId(     Integer.parseInt(parts[0])
);
    employee.setName(   parts[1]
);
    employee.setGender( parts[2].charAt(0)
);

employee.setAge(    Integer.parseInt(parts[3])
);

employee.setDuties( Integer.parseInt(parts[4])
);

    return employee;
  }

}
```

Question.java

```
/**
 * class: Question
 * purpose: Ask the user a question
 */

import java.util.Scanner;

public class Question {

  private String text;
  private String type;
  public int length;

  public Question (String _text, String _type)
{
    this.text = _text;
    this.type = _type;
    this.length = this.text.length();
  }

  /**
   * Ask question
   * @param length - length of text
   * @param scanner - Scanner instance
   * @return Object - answer
   */

  public Object ask (int length, Scanner
scanner) {

    Object input = null;

    this.print();

    if (type.equals("String")) {
      input = scanner.nextLine();
      if (((String) input).length() < 1) {
        System.out.println("Error! A value must
be entered.");
        return this.ask(length, scanner);
      }
    }

    else if (type.equals("int")) {
      while (! scanner.hasNextInt()) {
        scanner.nextLine();
        this.print();
      }
      input = scanner.nextInt();
      if (((Integer) input) < 0) {
        System.out.println("Error! Negative
values are not allowed.");
        return this.ask(length, scanner);
      }
    }

    else if (type.equals("gender")) {
      String text = scanner.next();
      if (text.length() == 0) {
        scanner.skip("\n");
        text = scanner.next();
      }
      input = text.charAt(0);
      if (input != 'm' && input != 'f') {
        System.out.println("Error! Please enter
'm' or 'f'.");
        return this.ask(length, scanner);
      }
    }

    else if (type.equals("yesno")) {
      String text = scanner.next();
      if (text.length() == 0) {
        scanner.skip("\n");
        text = scanner.next();
      }
      input = text.charAt(0);
      if (input == 'y') {
        input = true;
      } else if (input == 'n') {
        input = false;
      } else {
        System.out.println("Error! Please enter
'y' or 'n'.");
        return this.ask(length, scanner);
      }
    }

    return input;

  }

  /**
   * Ask question
   * @param scanner
   * @return Object
   */
  public Object ask (Scanner scanner) {
    return this.ask(text.length(), scanner);
  }


  /**
   * Get text
   * @return text
   */

  public String getText () {
    return this.text;
  }

  /**
   * Set text
   * @param text
   */

  public void setText (String _text) {
    this.text = _text;
  }

  /**
   * Get type
   * @return type
```

```
   */

  public String getType () {
    return this.type;
  }

  /**
   * Set type
   * @param type
   */

  public void setType (String _type) {
    this.type = _type;
  }

  /**
   * Print the question
   */

  private void print () {
    System.out.printf("%-" + length + "s ",
text);
  }

}
```

Questions.java

```
/**
 * class: Questions
 * purpose: Ask the user multiple questions
 */

import java.util.Scanner;

public class Questions {

  /**
   * Private variables
   */

  private int        maxLength;
  private Question[] questions;


  /**
   * Constructor
   * @param _questions - an array of questions
to ask
   */

  public Questions (Question[] _questions) {
    this.questions = _questions;
    this.findMaxLength();
  }


  /**
   * Ask the questions
   * @return Object[] - the answers
   */

  public Object[] ask (Scanner scanner) {

    // Create array to hold answers
    Object[] answers = new
Object[this.questions.length];

    // Skip any newlines
    scanner.skip("\n");

    // Ask each question
    for (int i = 0; i < this.questions.length;
i++) {
      answers[i] =
this.questions[i].ask(this.maxLength, scanner);
    }

    return answers;
  }


  /**
   * Find the length longest question
   * @return int - the length of the longest
question
   */

  private int findMaxLength () {
    int max = 0;
    for (int i = 0; i < this.questions.length;
i++) {
      if (this.questions[i].length > max) {
        max = this.questions[i].length;
      }
    }
    this.maxLength = max;
    return this.maxLength;
  }

}
```

Table.java

```
/**
 * class: Table
 * purpose: Generate a table using symbols
 */

import java.util.Arrays;

public class Table {

  /**
   * Private variables
   */

  private int           width;
  private String[][]    rows;
  private TableColumn[] columns;

  private final String V_SEP  = "|";
  private final String CORNER = "+";
  private final String H_SEP  = "-";
  private final String NL     = "\n";
  private final String PAD    = " ";

  /**
   * Constructor
   * @param _columns - an arary of table
columns
   */

  public Table (TableColumn[] _columns) {
    this.columns = _columns;
    this.rows = new String[][] {};
  }


  /**
   * Alternative constructor
   * @param _columns - table columns
   * @param _rows - table rows
   */

  public Table (TableColumn[] _columns,
String[][] _rows) {
    this(_columns);
    this.addRows(_rows);
  }


  /**
   * Print all columns
   * @return String - the table contents
```

```
   */

  public String print () {
    int[] index = new int[this.columns.length];
    for (int i = 0; i < index.length; i++) {
      index[i] = i;
    }
    return this.print(index);
  }


  /**
   * Print only certain columns by name
   * @param String[] - an array of the names of
each columns
   * @return String - table contents
   */

  public String print (String[] _columns) {

    String name;
    int[] index = new int[_columns.length];

    // Find out the index of each column
    for (int i = 0; i < this.columns.length; i+
+) {

      name = this.columns[i].getName();

      for (int j = 0; j < _columns.length; j++)
{

        if (name.equals(_columns[j])) {
          index[j] = i;
        }

      }
    }

    return this.print(index);

  }

  /**
   * Print only certain columns by index
   * @param int[] - an array of the indexes of
each column
   * @return String - table contents
   */

  public String print (int[] index) {

    String out = "";
    String line = "";
    int width;
    TableColumn column;
    String[] row;

    // Create the line
    // +----+---------+------+

    for (int i = 0; i < index.length; i++) {

      column = this.columns[index[i]];
      line += CORNER;
      width = column.getWidth();
      width += PAD.length() * 2;
      for (int j = 0; j < width; j++) {
        line += H_SEP;
      }
      if (i == index.length - 1) {
        line += CORNER;
      }
    }

    line += NL;

    // Top line
    out += line;

    // Left border and padding
    out += V_SEP + PAD;

    // Table column heading
    for (int i = 0; i < index.length; i++) {
      column = this.columns[index[i]];
      out += String.format(
        "%-" + column.getWidth() + "s",
        column.getName()
      );
      out += PAD + V_SEP + PAD;
    }

    // New line
    out += NL + line;

    // Print all rows
    for (int i = 0; i < this.rows.length; i++)
{
      row = this.rows[i];

      out += V_SEP + PAD;

      // Print columns defined by index
      for (int j = 0; j < index.length; j++) {
        column = this.columns[index[j]];
        out += String.format(
          "%-" + column.getWidth() + "s",
          row[index[j]]
        );
        out += PAD + V_SEP + PAD;
      }

      out += NL;
    }

    if (this.rows.length > 0) {
      out += line;
    }

    out += NL;

    System.out.print(out);
    return out;

  }


  /**
   * Add multiple rows to the table
   * @param _rows - table rows as a
multidimensional array of strings
   * {
   *   { "cell A1", "cell B1", "cell C1" },
   *   { "cell A2", "cell B2", "cell C2" },
   *   { "cell A3", "cell B3", "cell C3" },
   * }
   */

  public void addRows (String[][] _rows) {
    // Concatenate two arrays
    String[][] result =
Arrays.copyOf(this.rows, this.rows.length +
_rows.length);
    System.arraycopy(_rows, 0, result,
this.rows.length, _rows.length);
    this.rows = result;
  }

}
```

TableColumn.java

```
/**
 * class: TableColumn
 * purpose: Model a column of a talbe
 */

public class TableColumn {

  /**
   * Private variables
   */

  private int width;
  private String name;

  /**
   * Default constructor
   */

  public TableColumn () {
    this.width = 0;
    this.name = "";
  }

  /**
   * Alternative constructor
   * @param name - name of the column
   * @param width - width of the column
   */

  public TableColumn (String name, int width) {
    this.width = width;
    this.name  = name;
  }

  /**
   * Get the width of the column
   * @return int - the width
   */

  public int getWidth () {
    return this.width;
  }

  /**
   * Set the width of the column
   * @return int - the width
   */

  public void setWidth (int width) {
    this.width = width;
  }

  /**
   * Get the name of the column
   * @return String - the name
   */

  public String getName () {
    return this.name;
  }

  /**
   * Set the name of the column
   * @param String - the name
   */

  public void setName (String name) {
    this.name = name;
  }

}
```

Title.java

```
/**
 * class: Title
 * purpose: Display a box with centered text
 */

public class Title {

  /**
   * Private variables
   */

  // Width of the box
  private final int WIDTH = 64;

  private String title;
  private Table table;


  /**
   * Default constructor
   */

  public Title () {
    this.title = "";
  }


  /**
   * Alternative constructor
   * @param _title - the title of the table
   */

  public Title (String _title) {
    this.setTitle(_title);
  }


  /**
   * Print the table
   */

  public void print () {
    this.table.print();
  }


  /**
   * Set the title
   * @param _title - The title of the table
   */

  public void setTitle (String _title) {
    this.title = "# " + _title + " #";
    this.createTable();
  }


  /**
   * Create a new table that centers the title
text
   * @return this.table
   */

  private Table createTable () {

    // Calculate amount of padding required
    int length = this.title.length();
    int padding = (WIDTH / 2) + (length / 2);

    // Use String.format to prepend extra
spaces to the title
    String text = String.format("%" + padding +
"s", this.title);

    // Create a new table with the padded text
    this.table = new Table(new TableColumn[] {
      new TableColumn(text, WIDTH)
    });

    return this.table;
  }

}
```

TUI.java

```
/**
 * class: TUI
 * purpose: The Textual User Interface (TUI)
handles the input and output of the system.
This includes displaying menus, and lists as
well as listening to user input.
 */

import java.util.Scanner;

public class TUI {

  /**
   * Private variables
   */

  private Scanner scanner;
  private Employees employees;

  private final String MESSAGE_WELCOME =
"\nWelcome to the ABC Company Employee System\n
\n";
  private final String MESSAGE_NUMBER = "Number
of employees in system: ";
  private final String MESSAGE_FULL = "\nThe
maximum amount of employees has been reached.
\n";
  private final String MESSAGE_WAIT = "\nPress
ENTER to continue...\n";
  private final String MESSAGE_THANKS = "Thank
you for using the ABC Company Employee System
\n";
  private final String MESSAGE_WRITE = "Number
of employee records written to employees.txt
file: ";
  private final String NL = "\n";

  private final String ERROR_NO_RECORDS =
"Error! No employee records in the system.\n";

  private final TableColumn[] TABLE_EMPLOYEE =
{
    new TableColumn("ID", 3),
    new TableColumn("Name", 20),
    new TableColumn("Gender", 6),
    new TableColumn("Age", 3),
    new TableColumn("Number of Duties", 20)
  };

  private final Question[] QUESTION_EMPLOYEE =
new Question[] {
    new Question("Enter name:", "String"),
    new Question("Enter gender (m/f):",
"gender"),
    new Question("Enter age in years:", "int"),
    new Question("Number of jobs assigned:",
"int"),
  };

  /**
   * Default constructor
   */

  public TUI () {
    this.scanner = new Scanner(System.in);
    this.employees = new Employees();
    if (this.employees.fileExists()) {
      this.employees.read();
    }
    this.menu();
  }

  private void menu () {

    final TableColumn[] columns = {
      new TableColumn("#", 1),
      new TableColumn("Main Menu", 60)
    };

    final String[][] rows = {
      new String[] { "1", "Add employee" },
      new String[] { "2", "Delete an
employee" },
      new String[] { "3", "Modify an employee
record" },
      new String[] { "4", "List all
employees" },
      new String[] { "5", "View details of an
employee" },
      new String[] { "6", "List all employees
in an age group" },
      new String[] { "7", "View the employee
with the highest number of jobs assigned" },
      new String[] { "8", "Show total number of
employees with no jobs assigned" },
      new String[] { "9", "Exit" }
    };

    final Table table = new Table(columns,
rows);

    this.print(MESSAGE_WELCOME);
    this.numberOfEmployees();
    table.print();

    final Question prompt = new
Question("Select Option: ", "int");
    int selection = (Integer)
prompt.ask(this.scanner);

    switch (selection) {
      case 1: // Add employee
        this.add();
        break;

      case 2: // Delete employee
        this.delete();
        break;

      case 3: // Modify employee
        this.modify();
        break;

      case 4: // List all employees
        this.listAll();
        break;

      case 5: // View single emplyoee
        this.listSingle();
        break;

      case 6: // View age group
        this.listAgeGroup();
        break;

      case 7: // Most duties
        this.showMostDuties();
        break;

      case 8: // No duties
        this.showNoDuties();
        break;

      case 9: // Delete an employee
        this.exit();
        break;

    }

    // Show the menu again
    this.waitForUser();
    this.menu();

  }

  /**
   * 1. Add Employee
```

```
   */

  private void add () {

    final Title title = new Title("1. Add
Employee");
    title.print();

    // Check employees record is not full
    if (this.employees.isFull()) {
      this.print(MESSAGE_FULL);
      return;
    }

    Employee employee = new Employee();

    Questions questions = new
Questions(QUESTION_EMPLOYEE);
    Object[] answers =
questions.ask(this.scanner);
    this.setEmployee(employee, answers);

    int id = this.employees.add(employee);
    this.print("\nEmployee ID is " + id +
"\n");

  }


  /**
   * 2. Delete an employee
   */


  private void delete () {

    final Title title = new Title("2. Delete
Employee");
    title.print();

    final Question qId = new Question("Enter
Employee ID: ", "int");
    int id = (Integer) qId.ask(this.scanner);

    if (this.employees.has(id) == false) {
      this.print("Could not find employee " +
id + NL);
      return;
    }

    Employee employee = this.employees.get(id);
    this.printEmployeeDetails(employee);

    final Question qConfirm = new Question("Are
you sure you want to delete this record? (y/
N)", "yesno");
    boolean confirm = (Boolean)
qConfirm.ask(this.scanner);

    if (confirm == true) {
      this.employees.remove(id);
      this.print(NL + "Employee " + id + " has
been deleted." + NL);
    } else {
      this.print(NL + "Employee " + id + " has
NOT been deleted." + NL);
    }
  }


  /**
   * 3. Modify an employee
   */

  private void modify () {

    final Title title = new Title("3. Modify
Employee");
    title.print();

    final Question qId = new Question("Enter
Employee ID: ", "int");
    int id = (Integer) qId.ask(this.scanner);

    // Check employee ID exists
    if (!this.employees.has(id)) {
      this.print("Error - invalid ID" + NL);
      return;
    }

    // Get employee to be modified
    Employee employee = this.employees.get(id);

    // Ask questions
    Questions questions = new
Questions(QUESTION_EMPLOYEE);
    Object[] answers =
questions.ask(this.scanner);

    // Confirm changes
    Question confirm = new Question(
      NL + "Are you sure you want to modify
this record? (Y/n)",
      "yesno"
    );

    boolean confirmAnswer = (Boolean)
confirm.ask(this.scanner);

    if (confirmAnswer == false) {
      return;
    }

    // Modify employee
    this.setEmployee(employee, answers);
    this.print(NL + "Record modified." + NL);

  }



  /**
   * 4. List all employees
   */

  private void listAll () {

    final Title title = new Title("4. List of
All Employees");
    title.print();

    Table table = new Table(TABLE_EMPLOYEE,
this.employees.serialize());
    table.print();
    this.numberOfEmployees();
  }

  /**
   * 5. View single employee details
   */

  private void listSingle () {

    final Title title = new Title("5. Details
for a Single Employee");
    title.print();

    final Question qId = new Question("Enter
ID: ", "int");
    int id = (Integer) qId.ask(this.scanner);

    if (this.employees.has(id)) {

this.printEmployeeDetails(this.employees.get(id
));
    } else {
      this.print("Could not find an employee
with that ID." + NL + NL);
    }
```

```
    Question question = new Question("Would you
like to view another employee? (Y/n): ",
"yesno");
    boolean answer = (Boolean)
question.ask(this.scanner);

    // Keep running until the user enters 'n'
to stop
    if (answer) {
      this.print(NL);
      this.listSingle();
    }

  }

  /**
   * 6. View employees in an age group
   */

  public void listAgeGroup () {

    final Title title = new Title("6. Employees
in Age Group");
    title.print();

    Questions questions = new Questions(new
Question[] {
      new Question("Minimum age:", "int"),
      new Question("Maximum age:", "int")
    });

    Object[] age = questions.ask(this.scanner);

    Employees list = this.employees.inAgeGroup(
      (Integer) age[0],
      (Integer) age[1]
    );

    Table table = new Table(TABLE_EMPLOYEE,
list.serialize());

    this.print(NL);
    table.print(new String[] {
      "ID", "Name", "Age"
    });

    this.print(
      "\nThere are " + list.length +
      " employee(s) in the age range of " +
age[0] +
      " to " + age[1] + ".\n"
    );

  }

  /**
   * 7. Highest jobs assigned
   */

  public void showMostDuties () {

    final Title title = new Title("7. Employee
with Most Duties Assigned");
    title.print();

    if (this.employees.length > 0) {
      Employee employee =
this.employees.withMostDuties();
      this.printEmployeeDetails(employee);
    } else {
      this.print(ERROR_NO_RECORDS);
    }

  }

  /**
   * 8. No jobs assigned
   */

  public void showNoDuties () {

    final Title title = new Title("8. Employees
with No Duties Assigned");
    title.print();

    Employees list =
this.employees.withNoDuties();
    Table table = new Table(TABLE_EMPLOYEE,
list.serialize());
    table.print();

    this.print(
      "\nThere are " + list.length +
      " employee(s) with no jobs assigned.\n"
    );

  }


  /**
   * 9. Exit
   */

  private void exit () {

    final Title title = new Title("9. Exit");
    title.print();

    this.write();
    this.print(MESSAGE_THANKS);
    this.waitForUser();
    System.exit(0);
  }

  /**
   * Change employee settings
   */

  private void setEmployee (Employee employee,
Object[] answers) {
    employee.setName(   (String)
answers[0] );
    employee.setGender( (Character)
answers[1] );
    employee.setAge(    (Integer)
answers[2] );
    employee.setDuties( (Integer)
answers[3] );
  }

  /**
   * Write data to file
   */

  private void write () {
    this.employees.write();
    this.print(MESSAGE_WRITE +
this.employees.length + NL + NL);
  }

  /**
   * Print the number of employees in the
system
   */

  private void numberOfEmployees () {
    this.print(MESSAGE_NUMBER +
employees.length + NL + NL);
  }

  /**
   * Print the employee details
   * @param employee
   */

  private void printEmployeeDetails (Employee
employee) {
    String[][] rows = {
      employee.serialize()
    };
    Table table = new Table(TABLE_EMPLOYEE,
rows);
```

```
    table.print();
  }

  /**
   * Wait for the user to press enter
   */

  private void waitForUser () {
    this.print(MESSAGE_WAIT);
    this.scanner.skip(NL);
    this.scanner.nextLine();
  }

  /**
   * Print text to the screen
   * @param String - text to print
   */

  private void print (String string) {
    System.out.print(string);
  }


}
```

TUIStart.java

```
/**
 * class: TUIStart
 * purpose: Initialize the TUI
 */

public class TUIStart {

  public static void main (String[] args) {
    new TUI();
  }

}
```